Security Lab – Hacking-Lab Challenges Part 3

Introduction

In this lab, you'll do several Hacking-Lab challenges in the context of command injection, username enumeration, and cross-site scripting (XSS). The goal is to better understand and gain practical experience with such attacks. To access the challenges, select event *Challenges Part 3* in the Hacking-Lab – this shows you the seven challenges that are part of this event. It's recommended to work through the challenges in the given order.

1 Web Attack 3: Command Injection

1.1 Goal

Exploit a command injection vulnerability in the *Pinger* functionality of a simple web application to find a secret flag.

1.2 Required solution for full points

- A description of the attack steps you performed to find the secret flag.
- The secret flag.

1.3 Hints

- It appears that only the host *127.0.0.1* (localhost) can be pinged. Therefore, it's best to always use this IP address.
- Once you have managed to find and exploit the vulnerability in principle, have a look at the web application files for hints about the location of the secret flag…

2 Web Security: Username Enumeration

2.1 Goal

Exploit a username enumeration vulnerability in the *Galactic Empire Messenger* web application to get valid usernames, and identify the username of the account with password *DarkSide2021*. Note that getting valid usernames is in general identified as *username enumeration*.

2.2 Required solution for full points

- A description of the attack steps you performed to get valid usernames.
- All valid usernames you identified, and the username that corresponds to the account with password *DarkSide2021*.

2.3 Hints

- Use the list of username candidates that is provided in the challenge description.
- To do the attack, it may be helpful to use the *Intruder* component of *Burp* (which is rate-limited and slow in the available Community version, but it still can be used well to solve this challenge). Alternatively, you can use the *Fuzz* functionality of *OWASP ZAP* (which is very fast).

3 GlockenEmil 2.0 – XSS

3.1 Goal

Exploit a stored XSS vulnerability in the *Rate Product* functionality to steal the authentication token (stored in local storage with key *token*) of other users. When a victim views the products, the token should be sent to the attacker. This and further challenges use the *GlockenEmil 2.0* shop, which is a modern single page application that uses such authentication tokens instead of cookies. Note that more

details about authentication tokens and general information about the security of single page applications will follow later in the module SWS1. For now, you don't have to know about these details.

The credentials for the *GlockenEmil 2.0* shop are as follows (username/password): *customer0*/*compass0*, *customer1*/*compass1*,...

This challenge (and others that will follow) uses a second resource, a *Request Catcher*. This can be used by the attacker to capture stolen information such as cookies and authentication tokens. The URL *https://xyz.idocker.vuln.land/debug* is used to view the captured requests. Any other URL (e.g., *https://xyz.idocker.vuln.land/abcd*) stores the full request.

To carry out the attack, the victim would have to load the malicious JavaScript code in the browser, which sends the authentication token to the request catcher of the attacker. You can simulate this by using a second browser session, and logging in as a different user (e.g., *customer1/compass1*). This requests and executes the malicious JavaScript code.

3.2 Required solution for full points

- The JavaScript code you entered in a product rating to execute the attack.
- The full request (including all headers) captured by the request catcher when the attack is executed. This request must include the authentication token captured from the victim.

3.3 Hints

- To get to the *Rate Product* functionality, you must first log in (e.g., with *customer0/compass0*), then click *Shop* at the top, and next *Rate* to rate a product.
- Accessing the value of an object stored in local storage can be done in JavaScript with *localStorage.getItem('key')*, where *key* identifies the key of the object.
- Use the Firefox *Web Developer Tools* to inspect the content of the local storage.
- The request catcher can process GET and POST requests. Correspondingly, your JavaScript code to do the attack can use a GET or a POST request to send the authentication token to the request catcher. The easiest way is sending a GET request using the *fetch* API, as discussed in the lecture videos / slides.

4 XSS – DOM based

4.1 Goal

Find and exploit a DOM-based XSS vulnerability to get access to the cookie using JavaScript code. To access the web application, click the *DOM-based XSS* link on the entry page of the resource (you can ignore the part about *Theia Web IDE*). Once you have done this, you should see a web page with *Hacking-Lab for Fun and Profit* at the top. The vulnerability is located within this web page.

In contrast to the official challenge description that does not use a request catcher (and no corresponding resource to start), you'll solve this challenge using a request catcher. So simply reuse the one from the previous challenge or start a new one using the previous challenge description.

To carry out the attack, a prepared URL would be sent to the victim (e.g., in a link in an e-mail), which executes the malicious JavaScript code if the link is clicked, and which sends the cookie to the request catcher of the attacker. You can simulate this by using a second browser session, accessing the vulnerable web application (in the same way as described above), and copying the prepared URL into the address bar.

4.2 Required solution for full points

- A brief description of where on the web page the vulnerability is located.
- A URL, which can be entered in the address bar of the browser and that exploits the vulnerability by sending the cookie to the request catcher (the URL which you'd send to the victim).

- The full request (including all headers) captured by the request catcher when the attack is executed. This request must include the cookie captured from the victim.
- Explain whether or not it is possible to detect the attack at the server-side.

5 GlockenEmil 2.0 – DOM Local Storage

5.1 Goal

Find and exploit a DOM-based XSS vulnerability to steal the authentication token (stored in local storage with key *token*) of other users.

To carry out the attack, a prepared URL would be sent to the victim (e.g., in a link in an e-mail), which loads the malicious JavaScript code if the link is clicked, and which sends the authentication token to the request catcher of the attacker. You can simulate this by using a second browser session, logging in as another user (e.g., as *customer1/compass1*), and copying the prepared URL into the address bar.

5.2 Required solution for full points

- A URL, which can be entered in the address bar of the browser and that exploits the vulnerability by sending the authentication token to the request catcher (the URL which you'd send to the victim).
- The full request (including all headers) captured by the request catcher when the attack is executed. This request must include the authentication token captured from the victim.

5.3 Hints

- To get to the vulnerable resource, you must first log in (e.g., with *customer0/compass0*), and then click *Shop* at the top. The vulnerability is located in this resource.
- In case the JavaScript code you use does not work as intended, you may have to encode specific characters.

6 GlockenEmil 2.0 – SVG

6.1 Goal

Get access to the authentication token of another user (stored in local storage with key *token*) by exploiting a stored XSS vulnerability using a malicious SVG image. Such attacks are possible because SVG images can contain JavaScript code, which is executed when the image is rendered in the browser. The vulnerability is located in the *Community* area.

To carry out the attack, the victim would have to load the malicious SVG image in the browser, which sends the authentication token to the request catcher of the attacker. You can simulate this by using a second browser session, logging in as another user (e.g., as *customer1/compass1*), and selecting *Home* at the top to request the malicious image and to execute the malicious JavaScript code.

6.2 Required solution for full points

- A description of the attack steps you performed to get the authentication token. In particular, include the source code of the SVG image you used to do the attack.
- The full request (including all headers) captured by the request catcher when the attack is executed. This request must include the authentication token captured from the victim.

6.3 Hints

- To get to the vulnerable resource, log in, e.g., as *customer0/compass0*, and click *Community* at the top right. Here you can add a post that includes an SVG image.
- You can use the following code for a simple SVG image (place it in a file with ending *svg*):
  <?xml version="1.0" encoding="UTF-8"?>

```
<svg xmlns="http://www.w3.org/2000/svg" width="1000" height="1000" viewBox="0 0 32 32">
<rect fill="#f00" height="32" width="32"/><rect fill="#fff" height="6" width="20" x="6"
y="13"/>
<rect fill="#fff" height="20" width="6" x="13" y="6"/>
</svg>
```

- JavaScript code can be embedded anywhere in the SVG image, e.g., just before the closing *</svg>* tag.

7 SCHOGGI: Cross-Site Scripting (XSS) (Level 2)

Exploit a stored XSS vulnerability in the *Comment* functionality to steal the cookie of other users. When a victim views the comments, the cookie should be sent to the attacker. In this scenario, use username/password pairs *alice*/*alice.123* for the attacker and *bob*/*bob.123* for the victim. This challenge is not very different from some of the previous challenges and its main purpose is to learn that it is also possible to do XSS attacks without *<script>* tags (e.g., because such tags are filtered, or because the client-side framework prevents the execution of JavaScript code that uses *<script>* tags and that is dynamically inserted into the web page (the DOM) – which is for instance the case with Angular).

This challenge also uses a second resource, a *Request Logger*. It's basically a different variant of the previously used *Request Catcher* and works in a very similar way. The URL *https://xyz.idocker.vuln.land/x-log* is used to view the captured requests. Any other URL (e.g., *https://xyz.idocker.vuln.land/abcd*) stores the full request.

To carry out the attack, the victim would have to load the malicious JavaScript code in the browser, which sends the cookie to the request logger of the attacker. You can simulate this by using a second browser session, logging in as the victim (*bob*/*bob.123*), and clicking *MORE INFO* of the cake where the JavaScript code was inserted by the attacker. This requests and executes the malicious JavaScript code.

7.1 Required solution for full points

- The complete string (including the JavaScript code) you entered in a comment to execute the attack.
- The full request (including all headers) captured by the request logger when the attack is executed. This request must include the cookie captured from the victim.

7.2 Hints

- To get to the *Comment* functionality, you must first log in as attacker (*alice*/*alice.123*) and then click *MORE INFO* below one of the cakes.
- Enter the proof of concept JavaScript code *<script>alert("XSS");</script>* in a comment. As you can observe, the code is not executed, although you can assume that an XSS vulnerability actually exists.
- As an alternative, use *<img src='x' onerror='alert("XSS")'>* in another comment, which should execute the *alert* function. So apparently, it is indeed possible to exploit the XSS vulnerability by avoiding *<script>* tags. What happened behind the scenes? When rendering the comment in the browser, the image source is read from *'x'*, which is not a valid resource. This results in an error, and if the *onerror* attribute is specified, the corresponding JavaScript code is executed. With this knowledge, you should be able to solve this challenge.

Lab Points

In this lab, you can get **2 Lab Points**. To get them, you must achieve all 900 Hacking-Lab points that you can get from solving the challenges in this lab.